MACHINE LEARNING MODELS FOR DIRECTED CURATION OF DESIGN SOLUTION SPACE

by
Christian Huie Sjoberg

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Architecture

Charlotte

2017

Approved by:

_____
Eric Sauda

_____
Rachel Dickey

_____
Jefferson Ellinger

_____
Dr. John S. Gero

_____
Dr. Mirsad Hadzikadic

ABSTRACT

CHRISTIAN HUIE SJOBERG. Machine learning models for directed curation of design solution space. (Under the direction of ERIC SAUDA)

The expanding role of computational models in the process of design is producing exponentially growing parameter spaces. As designers, we must create and implement new methods for searching these parameter spaces considering not only quantitative optimization metrics but also qualitative features. This paper proposes a methodology leveraging pattern modeling properties of artificial neural networks to capture designer's inexplicit selection criteria and create user-selection based fitness functions for a genetic solver. Through emulation of learned selection patterns, fitness functions based on trained networks provide a method for qualitative evaluation of designs in the context of a given population. The application of genetic solvers for the generation of new populations based on the trained networks selections creates emergent high-density clusters in the parameter space, allowing for the identification of solutions which satisfy the designer's inexplicit criteria.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

GA	Genetic algorithm
ANN	Artificial Neural Network
MSE	Mean Square Error
RMSE	Root Mean Square Error
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
PCA	Principal Component Analysis

INTRODUCTION

As the influence of computation on design continues to grow, designers have the ability to optimize and account for high numbers of variables. This results in a high dimension solution space and an exponentially increasing number of possible variations. Optimization strategies enable designers to navigate toward solutions which are quantitatively superior in pre-defined tests, but limit the designer's ability to search the solution space for qualitative aspects. Fully accepting a quantitatively optimized solution poses an issue for designers. Calculating a top performing design based on limited or standardized criteria reduces the role of the designer. This has the potential to situate design as a secondary layer on calculated models. This workflow limits the ability of designers to holistically consider the aspects of design problems which are abstract or qualitative. To counteract this side-effect, designers may include quantitative descriptions of qualitative criteria into their computational models. In doing so, they are able to constrain the solution space based on their pre-conceived expectations of the possible outcomes. In either situation, the designer is limited in their exploration of the full solution space, and reduces the opportunity to discover emergent properties of designs resulting from unexpected combinations of parameters.

If we are to avoid this, we must re-evaluate the possible methods for navigating vast solution spaces and create models which can evaluate solutions based on both quantitative evaluations and qualitative properties. In the context of this project, the term qualitative is used to describe features of a computationally generated design which are visibly recognizable as favorable to the designer, but are not measured by the designer's script as an evaluation of the design's performance. These features can be understood to

be the emergent result of the combined state of many parameters of the design. Due to the abstract and subjective nature of qualitative criteria, designers can benefit from a process of selection of solutions based on their own expertise and intuition rather than the simplification of a problem and constraint of possible solutions.

We may imagine, for example, an artist in the process of making a sculpture. The artist does not explicitly or mathematically define the proper curvature of a surface of the sculpture before they begin to form it. They instead manipulate their design iteratively, and visually evaluate the current state of the form until they recognize that it has taken on the quality they find favorable. As humans, we are far better at visually recognizing a geometry we find to be favorable then we are at explicitly mathematically defining it. We may even find that our original assumptions or what we thought to be ideal changes in the process of creating a design.

The methods that computational designers currently employ to search for design options in a defined parameter space do not represent the inherent artistic processes of the designer, because they usually require the explicit definition of what is acceptable as a solution before the search process ever begins. If we can create software systems that more closely resemble the artist's evaluation of design qualities we can begin to remove some of the limitations that result from the need to explicitly define criteria at the beginning of our search. To perform meaningful evaluation of qualities subjectively favorable to an individual designer, a system must first create a model of the designer's evaluation process.

This project seeks to develop and test a method for searching vast solution spaces based on inexplicit evaluation criteria demonstrated by designers using the system.

Machine learning methods allow for the formation of evaluative criteria based on patterns learned from the user's selection of design options. As designers are iteratively presented with design choices, they will choose the solutions that meet their personal aesthetic, abstract, or qualitative criteria for design. The encoded design parameters for the possible selections as well as the actual user selections are recorded as a training set for an artificial neural network. A network trained on this selection data acts as a model of the user's evaluation of design options. Using this model as the fitness function for an evolutionary solver provides a mechanism for searching solution spaces containing more design possibilities than could possibly be evaluated individually by a human. Learning from the designer's evaluation of a sample of the possible solutions, this system demonstrates the ability to identify additional high performing regions of the solution space based on the user's inexplicit qualitative criteria. Allowing designers to visually identify the presence favorable qualitative features leverages the human's inherent strengths, while reducing the limitations resulting from the need for explicit limitation of the solution space.

A MACHINE LEARNING APPROACH

The discipline of architecture has explored the computational generation of designs through rule based logics for many years. One of the principle applications of this research has been the task of automated space planning. Charles Eastman's General Space Planner (1973) is an example of one such system for creating space planning designs using decision tree logic for evaluation. The application of multiple pre-defined constraints to the problem space is used as a method for searching for design solutions (Eastman 1973). These and other similar methods rely heavily on the user and system's

anticipation of all possible design conditions, and the presence of a pre-programmed response. This top down logic ultimately falls short of representing a human's process of design, which is one of nuanced reasoning, accumulated experience, and personal intuition. To enable a system to better represent the complex process of design, it must be developed to recognize subtle decisions as well as the greater patterns of designers. The application of a bottom-up approach to learning design constraints has the potential to remove the requirement of explicitly defined design constraints and better represent the designer's own process. It must be able to form a model of design responses that are not directly related to one specific condition, but to the presence of subtle patterns or combinations of multiple conditions.

The explicit curation of the solution space based on rules which operate at the level of the design parameter dramatically reduces the possibility of discovery of new options. Design is often a process of experimentation in which the designer tries numerous alternative approaches while both formulating and solving a problem. If the tool used by the designer requires them to explicitly define each of the criteria at the onset of the search for a solution, this process of exploration, discovery, and redirection is lost. To create tools which more naturally resemble the designer's process, we can begin to develop a bottom-up approach to the creation of constraints for the solution space. The collection of a designer's choices in their own process of curating possibilities to a design problem contains potentially identifiable patterns if we can consistently describe the steps of their process mathematically. Based on this limitation, a system capable of mathematically describing each of the design iterations explored by the designer can potentially form a model of the underlying relationships or patterns present in the set.

Creating an environment for the recording of the designer's process poses a difficult problem. We must make trade-offs between the freedom provided to the designer and the effectiveness of the system to measure the outputs of their process. Designers working with analog methods such as sketching and physical modeling produce artifacts which are difficult to comparatively evaluate, due to inconsistency in the relative meaning of any one determined variable between one artifact and the next. If, for example a series of drawings are compared which represent the designer's evolving understanding of a design, there is no guarantee that a specific pixel value is necessarily comparable with the same pixel in another image drawn at another scale, from another angle, or in another medium. For the purposes of this project, it is important to establish a consistent and comparable definition of the objects considered by the system.

The computational processes of architectural designers today are commonly developed within visual scripting platforms. These software platforms allow users with little to no programming experience to create models of the logic they wish to use to produce a design. This practice of scripting design demonstrates a voluntary, explicit definition of the possible parameter space. This project utilizes the user defined parameter space as a means of capturing consistent and comparable descriptions of the design options being considered. In this way, any potential design option within the user defined parameter space can be fully represented by its unique set of parameter values. This allows for simple comparison of one design's parameter sequence to another. The comparison of a series of designs selected by the designer for the presence of favorable qualitative traits provides the potential for pattern recognition between parameter values and user selection.

SEARCH METHODOLOGY

While these initial limitations create a finite parameter space, it is still potentially larger than a user could ever effectively search manually. To navigate these huge solution spaces, we must create a model of the user's selection criteria which the system can use in an automated search process. Artificial neural networks are effective at recognizing patterns in data when presented with numerous examples as a training set. Collecting a meaningful training set requires capturing the choices a user makes during the evaluation of design solutions. To achieve this goal, this project uses an iterative selection interface to allow for user evaluation. Users will be presented with a series of design options as a random sample of the dataset, and select the solutions they find most favorable from the set. This provides the system with a way of constraining the dataset to a specific design problem, as well as providing a weighted model of the user's choices. Object data from each user selected generation will be stored in the training set for the network. Given enough training objects, the neural network can begin to form a model of the user's design evaluation in the context of the possible design choices, and the user's top selections.

This tool will allow designers to search small percentages of vast possible solutions, capture their selection logic, and continue the search without the need for the designer's presence. Once this search is complete, the designer will be presented with the top results of the search. This allows for searches that are based on qualitative criteria that remain abstract to the search system.

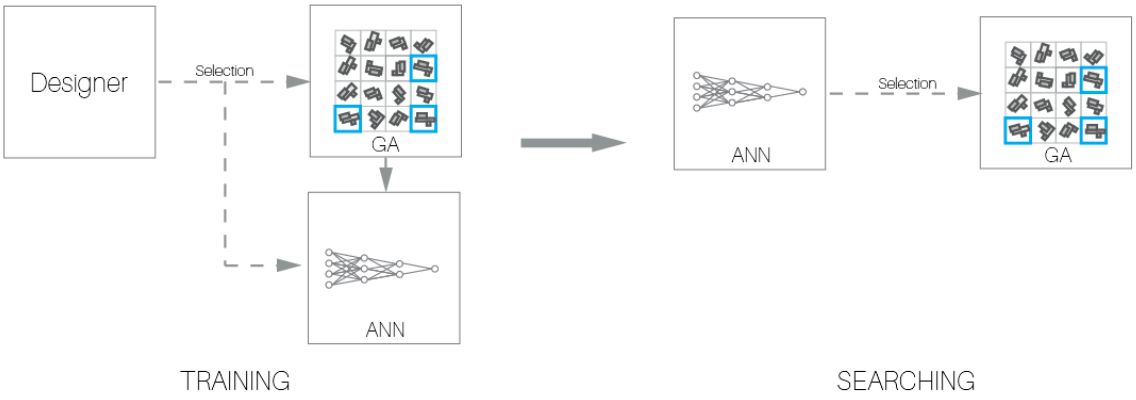


Figure 1: Diagram of training and search processes.

The trained network will act as a fitness function representing evaluation patterns in the user's selection process. During the user's selection process, they act as the fitness function for the GA by making the selections directly based on their own internal selection criteria which was unknown to the system. Once the system has created a model of this criteria by training the ANN, the selection process can be automated and the trained network can replace the user as the fitness function for the GA. Replacing the user with the trained ANN will provide the GA will have all components necessary to operate. The system will then be able to search the remaining options in the solution space one generation at a time using the ANN model of the user's selection criteria. During this process, the system will score each of the options it selects from each generation and return the highest scoring overall designs to the user. The design solutions which best match the user model will be returned by the system. Allowing the designer to make final selections from the curated solution set.

This project explores the pairing of artificial neural networks with genetic solvers as a means of searching solution spaces for learned selection criteria. Machine learning using artificial neural networks provides a framework for capturing relationships between

the design options a user is presented with and their selections of top performing designs. This selection model can then be used to iteratively search the remainder of the solution space by operating as a fitness function for a genetic solver.

BACKGROUND

The use of machine learning as a method of solving complex problems has been a reality of computer science since the 1950s. Early models of machine learning such as Arthur Samuel's model of the game of Checkers created in 1952 proved for the first time that a machine could learn to play a game better than its creator in a short period of time (Samuel 1957). This is considered to be the first machine learning algorithm developed. In the realm of design, it is conceivable that such a moment might also occur, where a system could eclipse the ability of human designers to consider and respond to the vast numbers of variables and relationships that influence design.

The writings and work of Nicholas Negroponte of MIT illustrate the recognition of this possibility within the design field. His work contemplated authorship in the era of machine aided design. Posing critical questions regarding the role that smart machines would play in design. He hypothesized that computing systems which he deemed "Architecture Machines" would one day go beyond the automation and repetition of tasks which are tedious to the architect. He also proposed that such systems would not only learn about architecture, but would "learn about learning about architecture." (Negroponte 1967) Negroponte's The Architecture Machine, lays out a framework for understanding the role of the computer in the process of design. He proposed that computers would operate as designers themselves, as partners to human designers, and as the physical environment. Understanding the role of computers as partners engaged in the process of design alongside human designers as opposed to simply as tools, can help us to consider new modes of interaction and communication of design intent. This early work

lead to the creation of The Architecture Machine Group at MIT and ultimately the formation of MIT Media Lab.

The contemplation of these levels of machine learning came shortly after the work of computer scientists such as Frank Rosenblatt, who explored the possibility of modeling the mechanisms of the brain and created the Perceptron. The Perceptron was based on a probabilistic model for information storage and organization in the brain (Rosenblatt, 1958). This algorithm used a two-layer model and was capable of recognizing patterns through the simple addition and subtraction of weights in the system. This development provided a mechanism for the creation of more complex models of machine learning. Multi-layer perceptrons became the model for artificial neural networks (ANN). These systems are capable of recognizing and forming a model of complex patterns when provided with training data (Hornik 1991). ANNs consist of interconnected layers of nodes. Each node in a layer receives values from nodes in the previous layer. These connections between nodes on different layers multiply the values that they pass by an initially random weight before sending them to the nodes on the next layer. Once a node receives these weighted values it normalizes the values by applying an activation function to the sum of all values it receives. The result of this activation function is then passed on to the next layer and this process repeats. The first and last layers in the ANN serve as the input and output of the algorithm. Training a network to understand the relationship between known pairs of input and output data is known as supervised learning (Love 2002). In order to train a supervised network, the network must be provided with the set of values for all nodes on the input and output layers. This pair of input and output values is known as a training object. Input node values are passed

through the network with the initially random connection weights. The difference between the values that the randomly weighted network outputs and the known true output values is then incrementally corrected for using a process called backpropagation. Backpropagation iteratively adjusts the weights between nodes in the network to correct the result at the output layer and make it more accurately match the known output value in the training object. (Hecht-Nielsen 1988) These corrections are made in small steps each time the network is given a new training object in the training set. When trained with a large enough set of training objects, the network can begin to form an accurate model of the relationship between inputs and outputs. Once this model is formed the ANN can be activated by providing it with only input values. Based on its model of the relationships in the training set, the trained ANN can return output values that match the given inputs. A trained network represents an approximated model of relationships in the provided training data. Designers and computer scientists began to consider the ways in which machine learning could be integrated into the design process.

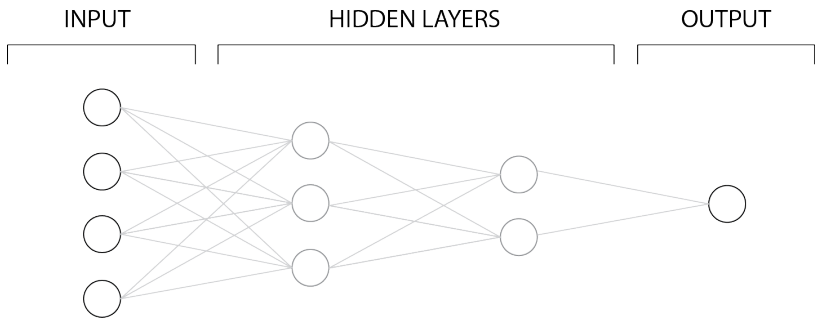


Figure 2: Structure diagram of a feed-forward artificial neural network

The proliferation of devices and information marked a shift in the field of machine learning as big data became more relevant. The ability to train on massive datasets and create complex models has given rise to the idea of "deep learning". This

phrase, coined by Geoffrey Hinton in 2006, describes a modern approach to machine learning which allows machine learning systems to "learn representations of data with multiple levels of abstraction" (LeCun et al. 2015). This describes machine learning models that use multi-layer ANNs and incorporate training methods to correct their models when presented with new information. Deep learning systems based on trained ANNs are effective at creating predictive models of complex phenomena. Modeling the complexity of design decisions with necessary levels of abstraction has previously been difficult or even impossible. Deep learning strategies provide a framework that designers must begin to explore in order to advance the ability to solve increasingly complex and high dimensional problems.

Searching solution spaces for desired metrics has become one of the most prominent methods of solving complex design problems in recent years. Black Box Studio at Skidmore, Owings, and Merrill (SOM) began exploring the ability to apply search algorithms to architectural optimization problems in the early 2000s. Their work focused on the use of genetic algorithms to search for optimized solutions to daylighting performance in tower massing (Besserud 2008). This work laid out the process for applying evolutionary genetic solvers to design problems. By identifying the parameters that could be manipulated in the form of the tower, they provided the algorithm with the genomes to manipulate. The fitness function or evaluation criteria for genomes in this study was based on the geometry's performance in a daylighting analysis. Using these inputs the genetic algorithm can iteratively create and test sets of individual design genomes and evaluate their performance. Each set, or generation, is created from the genes of the highest performing individuals in the previous generation. Although Genetic

algorithms are very effective at finding the single highest performing solution, they identify the goal of their work as the ability to visualize and evaluate a range of many well-performing design solutions (Besserud 2008). This work shows the increasing interest of designers and architects in the curation of possible design solutions through algorithmic methods. Recent projects by research groups at Autodesk have further explores the use of genetic algorithms as part of a "generative design" process. The application of multi-objective optimization algorithms to the task of space planning allows their system to balance six different criteria in the formation of a plan for an office space.(Nagy 2017)

A number of tools have become available for designers seeking to incorporate genetic algorithms in the architectural design process. Most notably Galapagos for Rhino Grasshopper (Rutten 2013) and Optimo for Revit Dynamo(Asl et al. 2015). These tools allow designers using visual scripting platforms to use genetic algorithms without needing to code entire process. The increasing complexity of models of design problems in architecture has furthered the exploration of methods for navigating vast solution spaces for pre-defined metrics, however current methods lack the ability to search for more abstract qualities that represent the designer's expertise or intuition. This project expands on the idea of search algorithms for finding design solutions by taking advantage of the strengths of genetic algorithms (GA) as search tools, and Artificial Neural Networks(ANN) for pattern recognition. An artificial neural network will be trained with a sample of design selections made by the user in the context of presented options. Once the network has been trained, it will emulate the designer and act as the fitness function for the genetic algorithm. (Figure 1) This will act as a mechanism for identifying design

options which best match the model of the user's selections. The application of machine learning methods for the formation of a model of the inexplicit evaluation heuristics of designers could enable searches of solution spaces to return design options representing the qualitative evaluation of the designer or designers making a selection.

IMPLEMENTATION

The implementation of this project is intended to accomplish two overarching goals: it must be able to train the network based on user selection, and perform a search of the remaining parameter space of the user-created script. In order to integrate this functionality into the workflow of architectural computational designers, this system is designed to be added to scripts created by Autodesk Dynamo users. The Dynamo environment acts as a front-end of the system and an external Python program executes the machine learning, evolutionary search, and cluster identification tasks. Results of the search of the parameter space are returned to the Dynamo interface to be presented to the user.

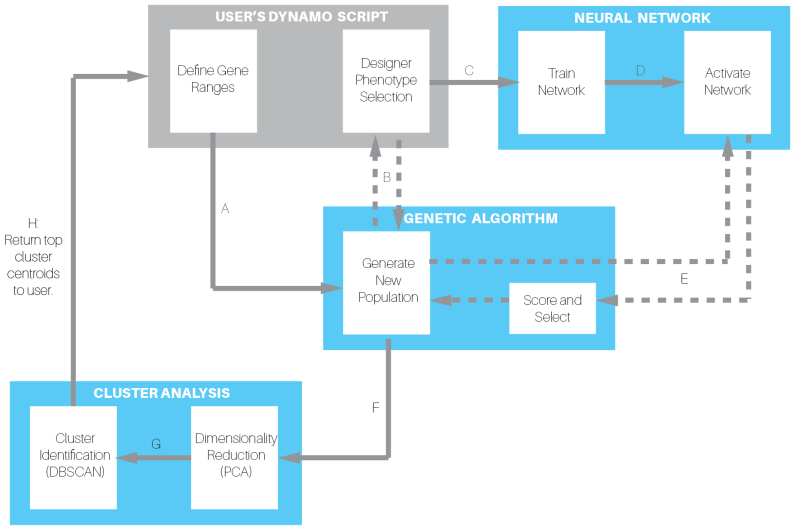


Figure 3: System Logic Diagram

Components in Autodesk Dynamo visually represent portions of code related to a specific functionality. The user provides the required input data streams on one side of the node, and receives the result of the components operations on the other side as output

data streams. Working within the dynamo environment allows users to append this system to the scripts, also known as graphs, which they have already developed as a means of searching for design solutions. The user defines all of the parameters and geometric constraints they wish to include in their design during the creation of their original Dynamo graph. Once the user has created their graph, they will add two additional components created for this project to aid in the training task.

Figure 4: Custom components created for Autodesk Dynamo visual scripting software.

USER SCRIPT SETUP

The first component, called GenomeEncoder, takes as input the range values for each parameter the designer wishes to include in the system. It is responsible for creating the random sample populations from which the user will make training selections, and recording their normalized parameters as a relative percentage of the range for each given parameter. This is accomplished by generating random parameter sequences within the range provided for each parameter of each design. The resulting two-dimensional list is then passed to the user-created portion of the graph which generates the geometric representation of the design options.

Visualization of the generated population of design options is left to the user. This allows the user to represent the design in whatever view or views they find suit the design. This gives the designer the freedom to represent buildings in plan, section, perspective or any other view type they wish to set up. Users may also wish to consider quantitative information about the designs such as graphing their efficiency, cost or other factors which help them to evaluate a design. Allowing the method of representation to be determined by the user mitigates the potential influence that a single prescriptive representation type may have on the user's evaluation of the design options. The Prototype design representation for this study used a simple array of aerial perspective renderings using standard Watch3d components in Dynamo. This method of visualization allows for simple comparison of the formal massing of the design options in a given population.

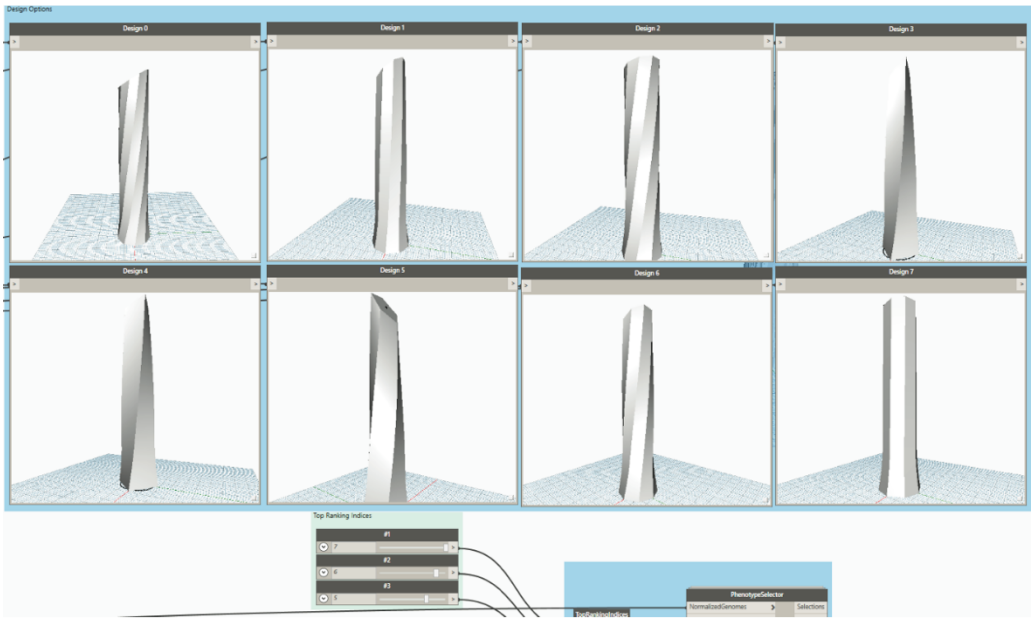


Figure 5: Example representation of generated design options in Autodesk Dynamo using standard Watch3D components.

The second component the user adds to their script is the PhenotypeSelector. This component takes in the normalized population parameter set from the GenomeEncoder, as well as the selection ranking values from the user, and assembles population-ranking pairs to be passed to the Python program for training. User selections from the sample population are provided as a list of the index numbers of the designs they identify as their top choices based on their personal criteria. Each time the Dynamo script is run, the selection and population data is aggregated in file in the directory provided by the user. The user iteratively makes selections and runs the Dynamo script as many times as they wish. The more selections the user makes, the more effective the system will be at determining the criteria they are using to select. Once the user feels they have made adequate selections from the sample populations, they can finish the training session by setting the final run input to “true”. The PhenotypeSelector will then export the training data to the Python program where it will later be used to train the machine learning algorithm and search the parameter space.

NEURAL NETWORK CONFIGURATION

Machine Learning algorithms can be described as either supervised or unsupervised systems. The primary difference being that in a supervised system both the input and expected output are provided during the training of the network, while in an unsupervised system only the inputs are provided.(Love 2002) Supervised machine learning offers some distinct advantages over unsupervised learning for this specific application. The capabilities of supervised artificial neural networks allow for a process which can be predictive as opposed to descriptive. Supervised machine learning can therefore be used to emulate of the way a designer might score a design option based on a

training set including other design populations and the design scores derived from user selection. Prediction of continuous value variables, such as a score value, are known as regression problems (Rasmussen 2006). Ultimately this creates the possibility of evaluating options from unexplored regions of the solution space based on evaluation patterns from a different region. This project uses a supervised feed-forward artificial neural network to create the predictive scoring model used for design evaluation.

The Python program created for this project relies heavily on the machine learning libraries PyBrain(Schaul et al. 2010) and Scikit-Learn(Pedregosa et al. 2011) to perform machine learning and data analysis tasks. The training set file generated by the PhenotypeSelector component serves as the primary input for the Python script. The raw population and ranking data in this file must be encoded properly before being fed to the network for training. Input-output pairs or 'training objects' must be assembled from each selection generation to create the final training set for the supervised network.

```
selectionPopulations = set of all populations the user was presented with
userSelections = set of rankings user gave to each individual in each population(1 is best)

For every population in selectionPopulations:
        For every design in the population:
                Get a list of all the design's parameters
                Add the parameter list to the input list

        For every selection in userSelections for that population:
                Get the rank of the selection
                If the rank is in the top three then:
                        Score = (1 – ((rank – 1)*.33))
                Otherwise score = 0
                Add score to the list of output values
```

Figure 6: Sudo code to create the final training set from populations shown to the user and their selections

The input side of each training object contains the sequence of parameter values that created the designs in the selection population. All inputs are normalized to a range

between zero and one representing their relative percentage along the parameters value range. This normalization prevents unequal weighting of parameters in the neural network. The output side of each training object contains the relative scores for the designs. Scores are calculated relative to the ranking of the design. The highest performing design receives a score of 1.00, the second receives a score of 0.66, the third receives a score of 0.33, and all other designs receive a score of zero. This creates a training object of input dimension equal to the product of the number of designs per generation and the number of parameters per design. The output dimension is equal to the number of designs per generation allowing for one score value per design. The network is then configured with the same input and output dimensions as the training objects and two hidden layers by default.

This network configuration ensures that scores are always presented to the network with their relative context. The network is being trained to evaluate designs in context of the presented options. This enables the network to later act as a fitness function for the genetic algorithm, iteratively evaluating the design options presented to it as inputs and returning their fitness scores relative to the population as outputs. This relationship is key to the searching capability of the system and its versatility in scoring regions of the parameter space outside of the initial training set based on its model of the user's scoring heuristics.

To provide a sufficient number of training objects for the network, the system can create an expanded training set containing all permutations of the order of individuals in each generation. Scores are also rearranged to match with their original design. This is done under the assumption that if the designer was shown the same exact population of

designs in a different order, their relative rankings in the population would remain the same. This process creates a training set of length equal to the number of generations times the factorial of the number of individuals in each generation. Using this method, very large datasets can be generated from relatively few user inputs. Although this method creates a more appropriately scaled training set for machine learning, it also has adverse effects on validation of the system since it produces a high chance of similar objects being used for both training and validation when the training set is split. It also dramatically increases training time for the network. For these reasons, this feature was disabled during user testing.

NEURAL NETWORK TRAINING

The configured network is then trained using the new training set. The backpropagation algorithm is used to train the network for multiple cycles or 'epochs' through the entire training set. Each epoch reinforces the relationships in the network by allowing the continued correction of connection weights in the network.(Hecht-Nielsen 1988) The number of epochs the network is trained for is determined by its performance on validation testing. Verification of the predictive capabilities of neural networks is commonly performed using the process of cross-validation.(Arlot 2010) Built-in PyBrain methods were used to perform cross-validation testing of the network. The system splits the training set into a training and validation set. In this case, 75 percent of the original training objects are used for training and 25 percent are used for validation. Using built-in PyBrain methods, the network is trained until it reaches a convergence between its ability to recreate the selection scores of the set its trained with and the ability correctly to predict the selection scores for the objects in the validation set. The accuracy of this

prediction is measured using the mean square error (MSE) between the systems predicted score and the actual user score. A MSE closer to zero represents a closer match between the predicted and actual. Generally speaking, MSE for the training set is lower than the validation set. This is due to the fact that the system has been exposed to the training set inputs already during training.(Sarle 1996)

One of the common issues faced in the training of neural networks is the possibility of overfitting. Overfitting occurs when the network is over trained or overly complex relative to the data it is used to predict. Overfitting results in poor predictive capability with inputs that were not part of the original training set (Dietterich 1995). Overfitting in the context of this project would potentially result in the system only recognizing the design options that the user selected during the training process as high scoring. This would then cause the system to only return options that were similar to the original user selections. Conversely, if the system is not sufficiently fit to the training set it will not be able to return designs which the user has identified as top performing. To avoid this issue, it is important that the system to have a balance between its ability to discover new solutions and return known high scoring designs. The PyBrain TrainUntilConvergence method used by this system helps to limit overfitting by selecting for training models which have the best balance between matching the training data and predicting the validation data.

The trained network is capable of being provided with the parameter values of all individuals in a population, and returning an array of score values representing the anticipated selection score for each individual. This network, when activated, is intended to represent the user's selection on the context of the input population.

SEARCHING THE PARAMETER SPACE

Once the model is trained, the system begins the search process. The search algorithm used by this system is a genetic algorithm. Genetic algorithms are a class of evolutionary solvers that perform an evaluation of individuals in a population using a fitness function, and generate new populations as a cross between the highest performing individuals, allowing some chance for mutation of gene values. This allows the algorithm to find the highest performing individuals in an overall population through a repeated process of evolution and selection. While the purpose of genetic algorithms is generally to identify the highest performing individual in a given population, it also can be modified to return every member of every population that was evaluated. This system analyses this aggregate of all generations searched by the algorithm as a method of identifying designs which meet the users exhibited criteria.

The search process must ultimately determine regions of high performing designs within the parameter space. To begin the search for these regions, an initial random sampling of the parameter space is created. All dimensions of the parameter space for the search have a range of zero to one. Matching the normalization of the parameters passed to the network. This sample contains the same number of individuals(designs) as the original training populations. If the user selected designs from a population of eight designs, and the network was trained to make selections from that data, then the size of the populations evolved by the genetic solver must also be eight individuals. This sample population is then formatted and provided to the trained neural network for scoring, and serves as the first generation of the genetic algorithm. The scores returned by activating the network with the sample generation provide the “fitness” for each of the individual

designs in the population. Using these fitness scores, the genetic algorithm evolves the population toward the highest performing scores. The parameters of the top scoring individuals in the population are crossed to create the next population.

As with any evolutionary solver, variation is introduced into the population through chance mutation of the genes, in this case design parameters, of the individuals. Mutation ensures that new regions of the parameter space are being sampled as new generations are evolved from the previous. As random mutations create higher performing designs, they are selected and the next generation will contain more of their traits. The results of this iterative search produce a set of generations of individuals from all regions of the multi-dimensional parameter space. Individuals returned from all evolutions of the population begin to form emergent structures in the parameter space. These structures are created by regions of high density. The concentration of individuals in a region of the parameter space is the result of high performance in the region. This is due to the ability of high performing individuals to pass their genes on to the next generation.

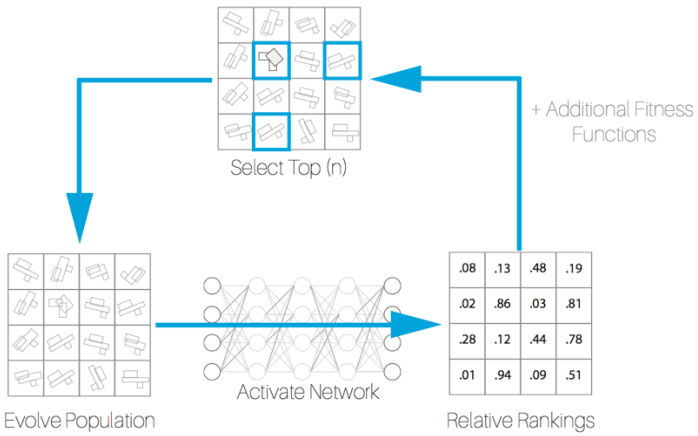


Figure 7: Search logic diagram showing the cycle of population evolution and scoring.

CLUSTER ANALYSIS

Identification of the parameter set at most dense regions of the parameter space determines the highest performing individuals found by the search. Evaluation of density is simple at lower dimensions where Euclidean distance calculations are reliable. Designs having high numbers of parameters create high dimension solution spaces, in which distance measurements are no longer reliable for determining the similarity between the parameter sets that define any two designs. At these higher dimensions, where more design parameters are considered, density evaluations based on distance suffer from what is known as *the curse of dimensionality.* As dimensions increase, similarity measurements become less meaningful.(Bellman, 1957) This results in reduced ability to effectively discover clusters in the data. To evaluate clusters of designs with high numbers of parameters, PCA (principal component analysis) is employed for dimensionality reduction. PCA is used to reduce higher dimension parameter spaces to a specified number of perpendicular summary dimensions.(Abdi et al. 2010) This system reduces the dimensions of the parameter space to three dimensions. A three dimensional PCA space allows for not only effective measurement of the similarity of design options, but also the ability to visualize the results of the search using a conventional scatter plot. This reduction in dimensionality makes clustering effective with higher numbers of design parameters.(Jolliffe 2002) Once the dimensionality reduction is completed and the parameter space can be effectively measured, the system must determine where clusters have formed in the PCA space.

Numerous algorithms exist for detecting regions of density in datasets, however not all can find a number of clusters that is not predetermined. To avoid arbitrarily

limiting the number of possible design solutions found by the system, the cluster analysis method used must not require a specific number of clusters to identify. In addition to this requirement, a clustering algorithm used by this system must be able to ignore noise in the space resulting from low performing random mutations. Mutations allow for the creation of designs which are dissimilar to the other members of the population, during the search process. If these individuals are selected for as higher performing than the current population, then they may begin to form a new cluster in their region of the parameter space. If they do not outperform the other members of the population, they become outliers in less dense regions of the parameter space. To minimize the potential effect of these low performance outliers on the process of identifying clusters, the clustering algorithm used by this system needs to be able to ignore these individuals. It must also be capable of finding non-spherical clusters since densities often occur in elongated bands connecting nearby regions of higher density. In studies of this system using three dimensional solution spaces, elongated bands of high density occur along the intersection of favorable parameter values. These bands of density cannot be identified by many clustering algorithms, since the average distance between the individuals are not consistent.

DBSCAN (*Density-Based Spatial Clustering of Applications with Noise*) developed by Martin Ester et. al (1996) meets these requirements and is used by this system to discover clusters. The DBSCAN algorithm requires the user to define an epsilon value which is effectively the minimum distance at which any two points are considered to be in the same “epsilon neighborhood”. Epsilon neighborhoods are used to create clusters of points which are each within a the epsilon distance from another

member of the cluster.(Ester 1996) In the context of design parameter space, this type of analysis is only effective if there is consistent, normalized parameter ranges. Without parameter normalization, the epsilon distance required to effectively find clusters would vary relative to the differences between parameter value ranges. The normalization used by this system guarantees that any parameter values will fall in the range zero to one. The resulting PCA space then has a consistent possible range of -1 to 1. This allows for a consistent value of epsilon regardless of the value ranges the user defines for each parameter in Dynamo. For the purposes of this initial study, an epsilon value of 0.04 is used for all cluster analysis performed. DBSCAN also requires a minimum number of members for a group of points to be considered a cluster. This value is set for the purposes of this project as 4% of the total number of individuals returned by the search process. The DBSCAN implementation used by this project(scikit-learn) returns the cluster label for each of the individuals in the analysis. Organizing the individuals into their respective clusters and ordering the clusters by their size allows the system to determine the most prominent clusters identified.

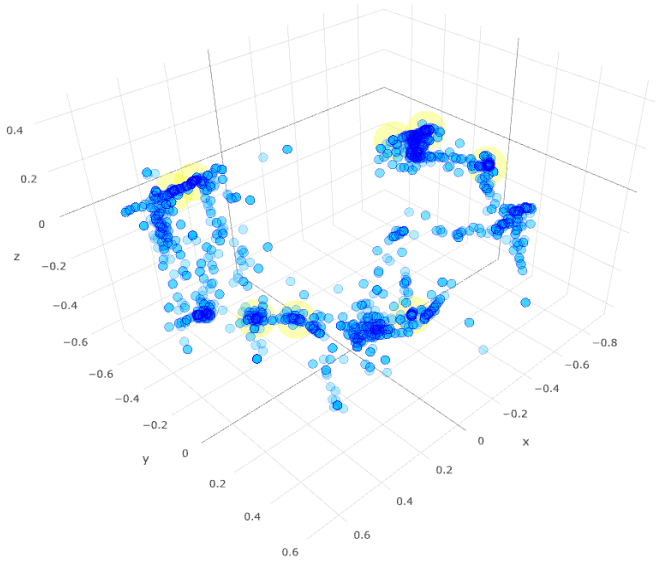


Figure 8: Identification of cluster centroids in 3D principle component analysis space using DBSCAN.

The centroid of each cluster can then be calculated to as the average of each parameter value for all individuals in the cluster. This centroid is used to represent the average design found the cluster. These centroids must be converted back to the original parameter space from the three dimensional PCA space before being returned to the user's Dynamo graph. This is accomplished by identifying the nearest neighbor to the centroid within the cluster. The index value of this individual can be used to reference the original higher dimension version of that individual design before dimensionality reduction was used. The design parameters must then be converted from their normalized ranges back to their original parameter value ranges from Dynamo. Once this conversion is complete, the set of design parameters are returned to the users Dynamo script where the parameter values can be used to re-create the geometric representations of the search results.

SYSTEM EVALUATION

A designer's intuitive and learned criteria for solving a design problem is both general and specific. It must be able to be applied to both new scenarios and scenarios they have designed for previously. Similarly, a robust machine learning model of their design criteria must be capable of evaluating options that were not part of its original training set. Testing of this system involved validation of the predictive capabilities of the model when shown new populations, as well as the user's acceptance of the systems selections as having met their inexplicit criteria.

STUDY PROCEDURES

Participants in the verification study for this system were asked to make selections of the top performing options from a set of color coded programmatic masses representing the schematic diagram of a building. Participants were not asked to develop the Dynamo script that produces this geometry. This parameter space was chosen to provide the complexity necessary to test the effectiveness of the PCA analysis while still producing geometric representations with recognizable characteristics which could be visually correlated with parameter values.

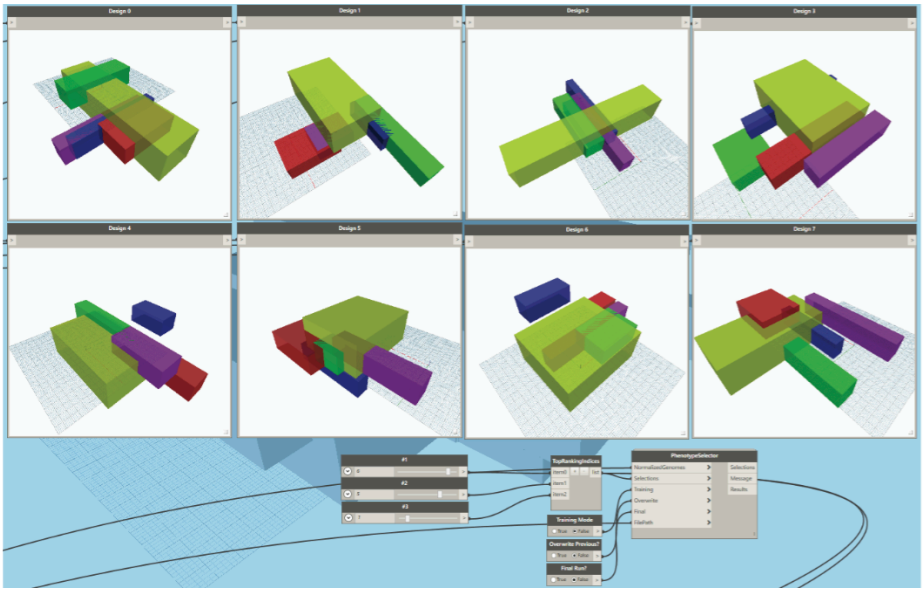


Figure 9: Random sample population of massing designs as seen by the user during the selection process.

Users were provided with three explicit base selection criteria to provide a means of gauging the system's ability to replicate the criteria. Any additional features they wished to consider could be included in addition to these criteria. The first group of participants were asked to rank designs based on how well they met the following design criteria:

- Program C (blue) must be adjacent or near program D (purple).
- Program A (yellow) must be on the ground level.
- Programs should have minimum overlap.

The first criteria, is intended to allow for the evaluation of the system's ability to recognize criteria resulting from a limited set of relative relationships between design parameters. In this case, the five parameters of both the blue and purple masses. The second criteria is intended to allow for the analysis of the system's ability to reproduce selections which have an absolute parameter value that is favored by the user. In this case, the z parameter of the yellow mass should have a value of zero when on the ground

plane. The third criteria tests the system's ability to produce results which satisfy a global relationship resulting from a combination of all parameters in the system. This requires the returned designs to satisfy different complex sets of possible combinations of parameters which result in this criteria being met.

Providing participants with well-defined criteria for selection is necessary in order to evaluate whether or not the system is able to emulate these criteria in its search process. The designer's selection criteria are not known to the system. These objectives were intended to produce consistency across the inputs in the study for the sake of comparison between users and also allow the system to be trained with the aggregate of all training objects produced by the participants. This produces a training set considerably larger than the system had previously been tested with. Each user was asked to make a minimum of 20 population rankings considering the three main selection criteria. Once the user completed their selection process, the system compiled the data and exported the training set to the neural network in Python. Training sets generated through this process were used to gauge the algorithms predictive capabilities. Quantitative evaluation of the systems predictive capabilities was conducted using cross-validation. Cross-validation performance was measured by the mean square error(MSE) of the system when predicting score values for populations in both the training and validation sets. Qualitative evaluation was performed through visual assessment of the systems returned designs for adherence to the design objectives provided to the users. This was also evaluated in the form of a questionnaire asking the participants to gauge the result sets based on their adherence to the design objectives.

CROSS-VALIDATION RESULTS

The ability of the training to converge on a model allowing for both replication and prediction of scores was primarily dependent on the user training set. All configurations and settings in the Python script were identical for each user. This helps to determine the variation brought on by the size of the training set and the consistency of the user's selections. For the purposes of this study, cross-validation was conducted using the aggregated training set produced by all seven participants on the study.

Figure 10 shows the MSE convergence during network training. MSE in the training set levels out at an average value of ~0.0153, while the validation set averages ~0.0928. This result shows that on average, the RMSE is ~0.1236 for the training set, and ~0.305 for the validation set. This variance does not account for the fact that selections are made based on the relative rankings of designs to others in the population. This correlation does however, show that even when trained with only 138 examples, the system can begin to emulate the designer's scoring.

Due to the relative highest score being selected for by the genetic algorithm, even a small correlation between the user's selections and the system's results in areas of density within the PCA space. Ultimately, these dense regions when found by the DBSCAN reveal areas of average higher scoring even with small datasets.

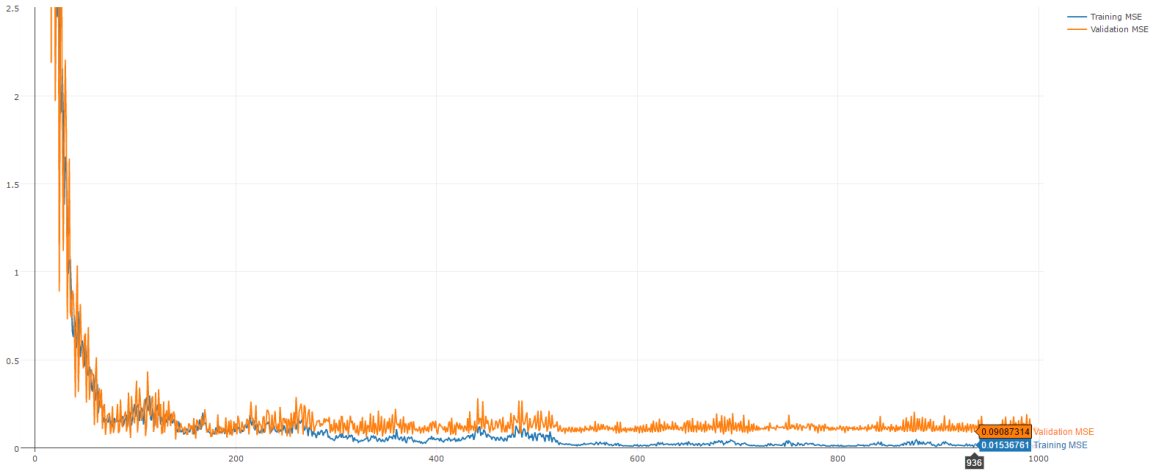


Figure 10: Training convergence MSE(Y) over training epochs(x).

OBSERVATIONS

There is no guarantee of similarity between the selection criteria of examples in the validation set vs the training set for the user when there are small numbers of training objects. The validation set could represent designs from a vastly different region of the solution space. In order for the system to create a meaningful model from relatively small datasets, the selections made by the user should follow some internal logic. The user's ability to select for multiple, possibly unrelated internal design criteria could result in a model that is very poor at predicting validation data. There is greater possibility of this occurring when the designer provides only a small training set to the network. In general, the system is more effective at forming a model of more abstract criteria with a larger training set. If the user only supplies a small training set, greater consistency in the user's selection criteria is required to return results with favorable traits. As the number of selections made by the user increases, the level of abstraction of selection rationale can increase as well.

Once the process of searching is completed, the participants in this study were asked to visually evaluate whether the designs found by the cluster analysis meet the general criteria provided. Of the seven participants involved in this study, four agreed that the returned designs matched the qualities of the design objectives, one strongly agreed, and one was neutral. Visual review of results returned to users suggests that the system can determine a generalized set of desired relationships between parameters in the design, however it did not perform as well at re-creating exact values for parameters which might be recognized by the user. For example, in most result populations the blue and purple masses are directly adjacent to each other, however the yellow mass does not always have an exact Z location of zero to place it on the ground plane. The yellow mass does however tend to maintain the relationship to the other masses that results from being on the ground plane, meaning that it is almost always the lowest volume in the set. The third design objective, to minimize overlap between masses, seems to also have some representation in the results returned, as most designs show a clustering of masses along an edge of the largest mass. This would suggest that the system has not been successful at meeting this criteria in the context of the other criteria.

To be able to definitively say that the criteria have been met, this study would have to be repeated with a larger group of participants. However, in comparison to the randomly generated starting conditions created by the system(Figure 9), the results returned do more closely meet the design criteria. These results show that this prototype system is capable of learning inexplicit criteria based on the users relative ranking of random sample populations from the solution space with small numbers of training objects. Larger datasets will likely result in dramatic improvement of the systems

recognition of criteria. Visual comparison of the PCA space graphs generated for each user show strong similarity in their relative regions of density. This suggests that the similarity in the user's criteria for selection is also represented in the systems evaluation of the solution space.

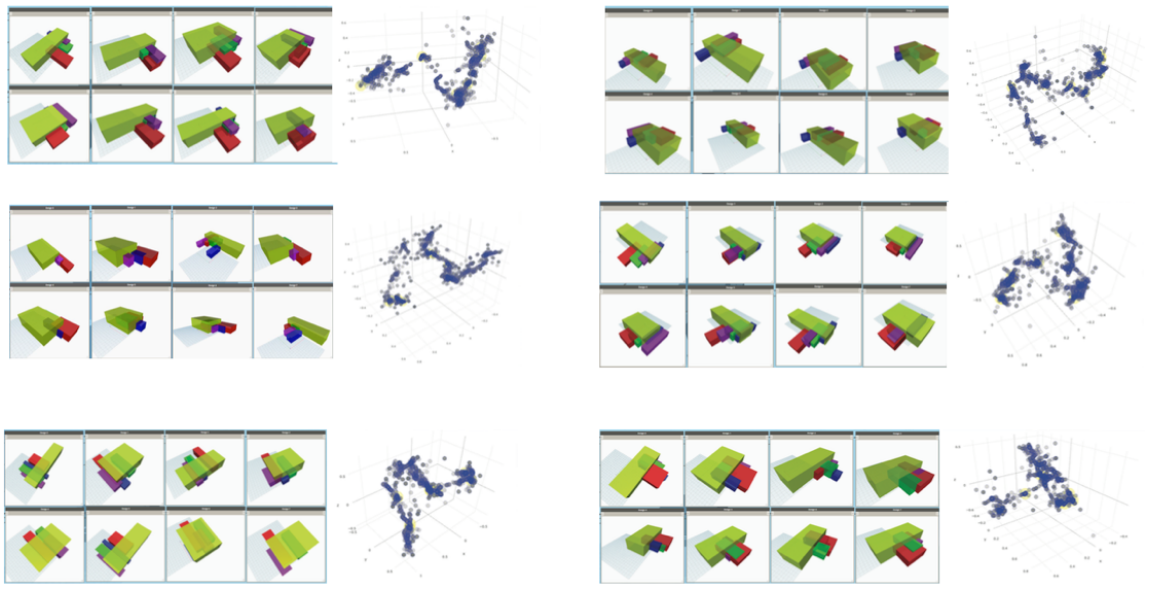


Figure 11: Massing designs returned by system after user training and search.

DISCUSSION AND FUTURE WORK

In order to better understand the implications of machine learning for design, this project has developed and tested a unique application of user-trained machine learning models as fitness functions for evolutionary solvers. This methodology allows for the emulation of inexplicit design criteria exhibited by users in a process of searching a parameter space. Identifying the most commonly selected individuals over the course of many generations using DBSCAN provides a method for curating the solution space for designs that best meet the criteria model. This is particularly applicable to the design process because this selection criteria is not bound to the specific samples of the solution space which were shown to the user. Due to the configuration of the training selection process, all rankings provided are relative to their specific generational context. This ultimately creates the opportunity for discovery of new design options outside of the limitations of the designs shown to the system during training.

Creation of a machine learning system that is general enough to be effective at forming a model of any number of criteria, and having any number of parameters is perhaps the greatest challenge in the development of this system. Configuration of the neural network based on the parameters of the design chosen by the user requires a highly extensible network configuration. As the user changes the number of variables in the designs, the network must scale in both size and complexity appropriately. Simple limited selection criteria may be best modeled by a network with only one or two hidden layers, while a designer wishing to use more abstract qualitative selection criteria may require a deeper network containing more hidden layers to form a model. Future work for this project will include the incorporation of optimization method for the configuration of the

network itself. The task of optimization of neural network architecture using evolutionary solvers has been explored by many researchers (Montana 1989). This improvement would likely increase the adaptability of the system to the user's specific training set.

Another challenge in the development of this system is the limitation of user selection through active input. Users of the system are not likely to spend long periods of time actively making selections. This results in a dataset that is extremely small relative to most applications of neural networks and machine learning. Small training sets cause several problems for the system including the propensity for overfitting and even greater limitations on the ability to validate predictive capabilities due to very small numbers of validation samples as a fraction of the training set. These limitations could be addressed by future development of more passive methods of collecting designer selection criteria. The architecture of this software is such that the means of generating the training set is left open, allowing for the possibility of scaling or modifying the user input process.

Creation of training sets has the potential to be a team or even crowdsourced task. Collaborative training by a group of designers or clients could provide a means of generating much larger and perhaps more meaningful training sets for accomplishing design tasks. Future work for this project will include a method of training using collaborative training sets. The parameterization of the designs by the system allows this system to potentially be used for any application in which a user is selecting from options which can be described numerically. Systems following this proposed methodology are particularly applicable to problems containing high numbers of variables and having visual or geometric representation able to be evaluated by the user.

Future work for this project will also explore the inclusion of trained networks into multi-objective optimizations for design. The increasing use of quantitative performance optimizations in the field of design has the potential to place the designers input as a secondary layer on optimized or engineered solutions. Qualitative evaluation functions based on trained neural networks provide a potential method for the integration of design knowledge into optimization systems used by architects and other designers today. By including learned design knowledge in our optimization processes, we can begin to create optimization systems which better represent the intent of the designers who use them.

CONCLUSION

This project implements a method for capturing and emulating designer's inexplicit qualitative selection criteria as a means of searching vast solution spaces for design options. The ability to evaluate and recognize abstract qualitative criteria through visual input is one of the human designers most fundamental skills. Neural networks are capable of modeling patterns identified by a designer's process of selection within a defined parameter space. The iterative search power of the machine when paired with trained neural network models for evaluation leverages the strengths of both the designer and the machine. This is a potentially powerful method of discovering designs favorable to the designer in solution spaces larger than the designer could ever thoroughly search. This work shows that even with small numbers of training objects, an algorithm with this configuration is capable of forming a model that begins to emulate the design criteria of the user who trained it.

REFERENCES

Abdi, Hervé, and Lynne J. Williams. "Principal component analysis." *Wiley interdisciplinary reviews: computational statistics* 2, no. 4 (2010): 433-459.

Arlot, Sylvain, and Alain Celisse. "A survey of cross-validation procedures for model selection." *Statistics surveys* 4 (2010): 40-79.

Asl, Mohammad Rahmani, Saied Zarrinmehr, Michael Bergin, and Wei Yan. "BPOpt: A framework for BIM-based performance optimization." *Energy and Buildings* 108 (2015): 401-412.

Besserud, Keith, and Joshua Cotten. "Architectural genomics." *Silicon+ Skin> Biological Process and Computation* (2008): 238-245.

Dietterich, Tom. "Overfitting and undercomputing in machine learning." *ACM computing surveys (CSUR)* 27, no. 3 (1995): 326-327.

Eastman, Charles M. "Automated space planning." *Artificial intelligence* 4, no. 1 (1973): 41-64.

Ester, Martin, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. "A density-based algorithm for discovering clusters in large spatial databases with noise." In *Kdd*, vol. 96, no. 34 (1996): 226-231.

Hecht-Nielsen, Robert. "Theory of the backpropagation neural network." *Neural Networks* 1, no. Supplement-1 (1988): 445-448.

Jolliffe, Ian. "Principal component analysis." John Wiley & Sons, Ltd, (2002): 214-219.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." *Nature* 521, no. 7553 (2015): 436-444.

Love, Bradley C. "Comparing supervised and unsupervised category learning." *Psychonomic bulletin & review* 9, no. 4 (2002): 829-835.

Montana, David J., and Lawrence Davis. "Training Feedforward Neural Networks Using Genetic Algorithms." In *IJCAI*, vol. 89 (1989): 762-767.

Nagy, Danil, Damon Lau , John Locke, James Stoddart, Lorenzo Villaggi, Ray Wang, Dale Zhao, David Benjamin. **"**Project Discover: An application of generative design for architectural space planning**"** *SimAUD 2017 Conference proceedings: Symposium on Simulation for Architecture and Urban Design* (2017):

Negroponte, Nicholas. "Toward a theory of architecture machines." *Journal of Architectural Education* 23, no. 2 (1969): 9-12.

Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel et al. "Scikit-learn: Machine learning in Python." *Journal of Machine Learning Research* 12, no. Oct (2011): 2825-2830.

Rasmussen, Carl Edward. "Gaussian processes for machine learning." (2006).

Rosenblatt, Frank. "The perceptron: A probabilistic model for information storage and organization in the brain." *Psychological review* 65, no. 6 (1958): 386.

Rutten, David. "Galapagos: on the logic and limitations of generic solvers." *Architectural Design* 83, no. 2 (2013): 132-135.

Samuel, Arthur L. "Some studies in machine learning using the game of checkers." *IBM Journal of research and development* 3, no. 3 (1959): 210-229.

Sarle, Warren S. "Stopped training and other remedies for overfitting." *Computing Science and Statistics* (1996): 352-360.

Schaul, Tom, Justin Bayer, Daan Wierstra, Yi Sun, Martin Felder, Frank Sehnke, Thomas Ruckstieß, and Jurgen Schmidhuber. "PyBrain." *Journal of Machine Learning Research* 11, no. Feb (2010): 743-746.

APPENDIX A: SUDO CODE FOR TRAINING SET ENCODING

Create training set from populations shown to the user and their selections:

```
selectionPopulations = set of all populations the user was presented with
userSelections = set of rankings user gave to each individual in the population

For every population in selectionPopulations:
        For every individual in the population:
                Get a list of all the individuals parameters
                Add the parameter list to the input list

        For every selection in userSelections:
                Get the rank of the selection
                If the rank is in the top three then:
                        Score = (1 – ((rank – 1)*.33))
                Otherwise score = 0
                Add score to the list of output values

        trainingObject = (input values, output values)
        add trainingObject to trainingSet
```

APPENDIX B: USER STUDY QUESTIONNAIRE

User Study Questionnaire Page1

Machine Learning Models for Directed Curation of Design Solution Space

Participant ID:_____ Date:_____

Please indicate how much you agree or disagree with the following statements by filling in the corresponding circle.

1) Scripting tools are useful for design.

○ Strongly Disagree ○ Disagree ○ Neutral ○ Agree ○ Strongly Agree

2) I use scripting tools often in my own design process.

○ Strongly Disagree ○ Disagree ○ Neutral ○ Agree ○ Strongly Agree

3) I have a high degree of scripting / programming experience for a designer.

○ Strongly Disagree ○ Disagree ○ Neutral ○ Agree ○ Strongly Agree

4) I frequently install additional add-ons, plug-ins, and packages for my scripting tools.

○ Strongly Disagree ○ Disagree ○ Neutral ○ Agree ○ Strongly Agree

5) I am experienced with the Autodesk Dynamo scripting environment.

○ Strongly Disagree ○ Disagree ○ Neutral ○ Agree ○ Strongly Agree

6) The design criteria I was given had easily comparable characteristics to the designs I was shown by the tool.

○ Strongly Disagree ○ Disagree ○ Neutral ○ Agree ○ Strongly Agree

7) The selections I made represented the qualities / characteristics of the criteria I was given.

○ Strongly Disagree ○ Disagree ○ Neutral ○ Agree ○ Strongly Agree

User Study Questionnaire Page 2

Machine Learning Models for Directed Curation of Design Solution Space

Participant ID:_____ Date:_____

Please indicate how much you agree or disagree with the following statements by filling in the corresponding circle.

8) Making selections based on my prompt criteria was simple.

○ Strongly Disagree ○ Disagree ○ Neutral ○ Agree ○ Strongly Agree

9) I understand the inputs / controls of this tool.

○ Strongly Disagree ○ Disagree ○ Neutral ○ Agree ○ Strongly Agree

10) The outputs of this tool matched the qualities of the designs I selected based on my criteria.

○ Strongly Disagree ○ Disagree ○ Neutral ○ Agree ○ Strongly Agree

11) I was able to identify the selections I made from the set shown.

○ Strongly Disagree ○ Disagree ○ Neutral ○ Agree ○ Strongly Agree

12) This tool would be useful for finding designs that meet my qualitative criteria in my own projects.

○ Strongly Disagree ○ Disagree ○ Neutral ○ Agree ○ Strongly Agree

13) I would spend time training a tool like this to find solutions to complex design problems in my own projects.

○ Strongly Disagree ○ Disagree ○ Neutral ○ Agree ○ Strongly Agree

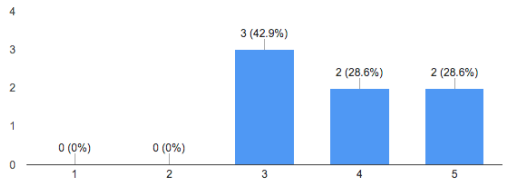
14) I use tools to search for solutions to design problems in my own work.

○ Strongly Disagree ○ Disagree ○ Neutral ○ Agree ○ Strongly Agree

APPENDIX C: USER STUDY QUESTIONNAIRE RESPONSES

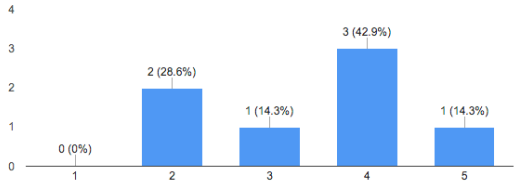
I use scripting tools often in my own design process.

7 responses



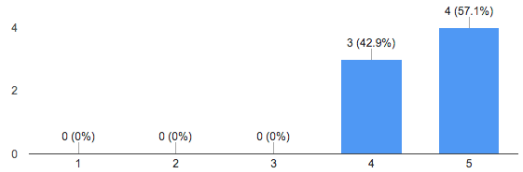
I have a high degree of scripting / programming experience for a designer.

7 responses



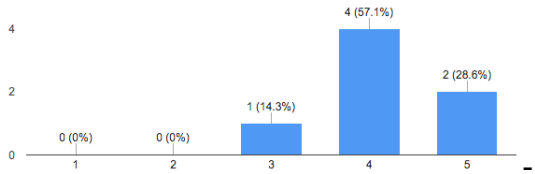
Scripting tools are useful for design.

7 responses



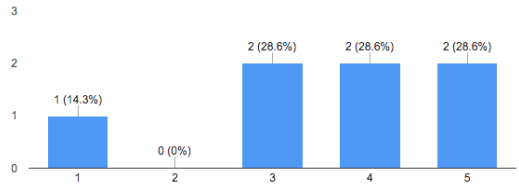
I am experienced with the Autodesk Dynamo scripting environment.

7 responses



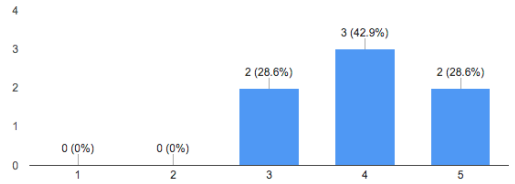
I frequently install additional add-ons, plug-ins, and packages for my scripting tools.

7 responses



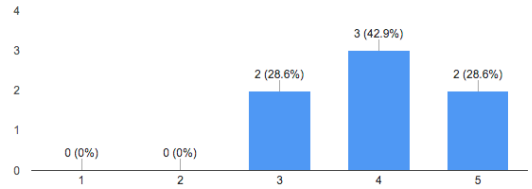
The design criteria I was given had easily comparable characteristics to the designs I was shown by the tool.

7 responses



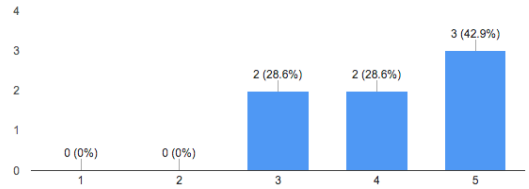
The selections I made represented the qualities / characteristics of the criteria I was given.

7 responses



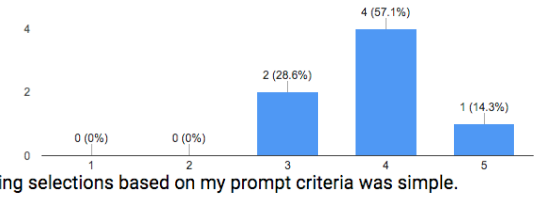
I understand the inputs / controls of this tool.

7 responses



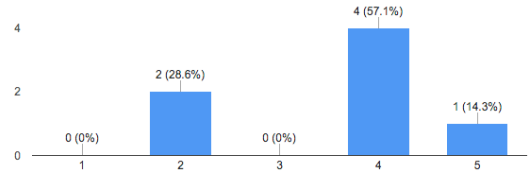
The outputs of this tool matched the qualities of the designs I selected based on my criteria.

7 responses



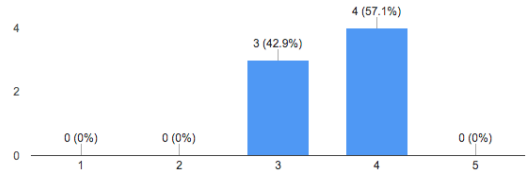
Making selections based on my prompt criteria was simple.

7 responses



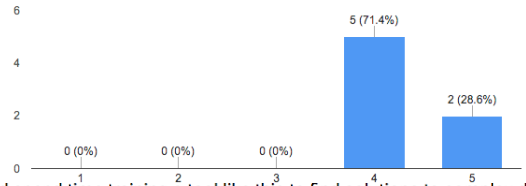
I was able to identify the selections I made from the set shown.

7 responses



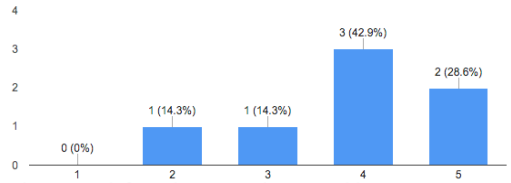
This tool would be useful for finding designs that meet my qualitative criteria in my own projects.

7 responses



I would spend time training a tool like this to find solutions to complex design problems in my own projects.

7 responses



I use tools to search for solutions to design problems in my own work.

7 responses